JavaScript

Tags

1 - CONDITION

BÀI 1:

Viết hàm lấy giá vé tương ứng theo số tuổi

Khi khách hàng vào tham quan khu du lịch, tuỳ thuộc vào độ tuổi mà có giá vé khác nhau.

Hãy viết một hàm nhận vào số tuổi của khách hàng, trả về giá vé tương ứng.

Bảng giá như sau:

- Dưới 6 tuổi hoặc từ 70 tuổi trở lên là được FREE (trả về 0)
- Từ 6 - 12 tuổi giá vé là 20.000 VND
- Trên 12 tuổi thì đồng giá 50.000 VND

Viết hàm `function getTicketPrice(age)` để giúp mình lấy được giá tiền khi biết tuổi của khách hàng nhé.

Trả về là một con số nguyên:

- 1 nếu dữ liệu không hợp lệ
- 0 nếu free
- các giá tiền nếu tuổi hợp lệ

Giả sử người sống thọ nhất là 125 tuổi.

BÀI 2

Say hello bằng nhiều ngôn ngữ

Viết hàm `sayHello(languageCode)` nhận vào **languageCode** và trả về câu xin chào của ngôn ngữ tương ứng.

- languageCode = 'en' --> Hello
- languageCode = 'vi' --> Xin chào
- languageCode = 'fr' --> *Bonjour*
- languageCode = 'cn' --> *Nĭn hăo*
- languageCode = 'ja' --> *Konnichiwa*
- languageCode = 'ko' --> *Anyoung haseyo*
- Trường hợp **không truyền languageCode** hoặc **languageCode không hợp lệ** thì trả về mặc định "**Hello**"

Link để copy các ngôn ngữ khác nhau: click here

Mọi người chú ý, copy các chuỗi trả về cho nó đúng nhé, đừng tự gõ nhen!

2 - NUMBER

💡 BÀI 1:

Bài toán gọi taxi

Viết hàm `getTaxiCount(passengersCount)` nhận vào là **số hành khách** và trả về là **số taxi cần gọi** để chở hết số khách đó.

Yêu cầu:

- Khi số khách lớn hơn 4, ưu tiên dùng xe 7 chỗ
- Nếu số khách nhỏ hơn hoặc bằng 4 thì dùng xe 4 chỗ

Trả về con số duy nhất là tổng của cả 2 loại xe taxi cần phải gọi.

Ví dụ:

- getTaxiCount(3) --> **1** xe 4 chỗ
- getTaxiCount(6) --> **1** xe 7 chỗ
- getTaxiCount(10) --> **2** xe vì 1 xe 7 chỗ và 1 xe 4 chỗ
- getTaxiCount(20) --> **3** xe 7 chỗ

Happy coding!

💡 BÀI 2:

Tìm chữ số lớn nhất của một số nguyên dương

Viết hàm `function getMaxDigit(n) {}` để tìm ra chữ số lớn nhất của một số nguyên dương **n** (0 <= n < 1000)

Ví dụ:

- getMaxDigit(1) --> 1
- getMaxDigit(12) --> 2
- getMaxDigit(123) --> 3
- getMaxDigit(921) --> 9

Trường hợp dữ liệu không hợp lệ, nằm ngoài vùng cho phép của **n** thì trả về **-1**

Lưu ý: không sử dụng vòng lặp vì chưa học kiến thức này.

BÀI 3:

Viết hàm `compareNumbers(a, b)` nhận vào 2 số nguyên **a, b** bất kỳ.

Trả về:

- 1 nếu a > b
- 0 nếu a = b
- 1 nếu a < b

Ví dụ:

- compareNumbers(3, 5) --> -1
- compareNumbers(9, 7) --> 1
- compareNumbers(7, 7) --> 0

Happy coding!

BÀI 4:

Kiểm tra số có tối đa 3 chữ số có phải là số đối xứng hay không?

Viết hàm `isSymmetricNumber(n)` để nhận vào số nguyên dương **n** có **tối đa 3 chữ số** và trả về **true nếu n là số đối xứng**, ngược lại trả về false.

Giả sử tham số **n** truyền vào luôn luôn là số có tối đa 3 chữ số. (0 --> 999)

Số đối xứng là số mà đọc từ trái sang phải nó giống như đọc từ phải sang trái.

Ví dụ: 1, 22, 33, 121, 222, 353, 373, ...

Lưu ý: không dùng chuỗi, không dùng mảng, chỉ xử lý bằng số thôi nhé.

3 - STRING:

💡 BÀI 1:

Tìm và xoá các nguyên âm trong câu văn

Viết hàm `function removeVowel(str)` để tìm và remove tất cả nguyên âm có trong câu văn đầu vào.

Nguyên âm là các ký tự: **u, e, o, a, i** (uể oải :P)

Ví dụ:

- removeVowel('') --> ''
- removeVowel('say hello') --> 'sy hll' vì **a, e, o** là nguyên âm nên đã bị xoá

Giả định là mỗi nguyên âm trong **str** chỉ xuất hiện một lần.

Trường
 hợp chuỗi sau khi bỏ các nguyên âm có dư khoảng trắng ở đầu hoặc cuối chuỗi, thì hãy bỏ luôn các khoảng trắng thừa này nhé.

Lưu ý: không được sử dụng for, chỉ được dùng hàm replace()

BÀI 2:

Format số giây luôn hiển thị 2 chữ số

Viết hàm `formatSeconds(seconds)` nhận vào là số giây **(0 <= seconds < 60)** và trả về chuỗi luôn có 2 chữ số của số giây.

Bài
 này áp dụng trong thực tế khi muốn show đồng hồ điện tử, để cho đẹp, mình đảm bảo luôn show 2 chữ số, dù số giờ phút giây chỉ có một chữ số.

Ví dụ:

- formatSeconds(0) --> '00'
- formatSeconds(9) --> '09'
- formatSeconds(20) --> '20'

Viết hàm theo 2 hướng tiếp cận khác nhau:

1. `formatSecondsV1(seconds)` Sử dụng if...else
2. `formatSecondsV2(seconds)` Sử dụng hàm slice()

Happy coding!

BÀI 3:

Convert số giây sang chuỗi hh:mm:ss

Viết hàm `formatTime(seconds)` nhận vào là số giây **seconds** với (0 <= seconds <= 86400)

Trả về là một chuỗi với định dạng **hh:mm:ss** trong đó:

- **hh** là số giờ
- **mm** là số phút
- **ss** là số giây

hh, mm, ss luôn luôn hiển thị 2 chữ số kể cả nhỏ hơn 10.

Ví dụ:

- formatTime(0) --> '00:00:00'
- formatTime(9) --> '00:00:09'
- formatTime(4256) --> '01:10:56'

Happy coding!

BÀI 4:

Kiểm tra URL có sử dụng phương thức bảo mật

Viết hàm `isSecureUrl(url)` nhận vào là một **url** và trả về **true nếu url có sử dụng phương thức bảo mật**, ngược lại trả về false.

Quy ước URL được xem là có sử dụng phương thức bảo mật nếu bắt đầu bằng:

- https
- wss

Tạm thời bỏ qua việc kiểm tra url có hoàn chỉnh hay không, chỉ quan tâm phương thức bảo mật.

Ví dụ:

- isSecureUrl('**http**://abc.com') --> false vì bắt đầu bằng http, ko phải là https
- isSecureUrl('**https**://ezfrontend.com') --> true
- isSecureUrl('**wss**://chat.ezfrontend.com') --> true
- isSecureUrl('**ws**://chat.abc.com') --> false

Viết bằng 2 hướng tiếp cận khác nhau:

1. `isSecureUrlV1(url)` Sử dụng **indexOf**()
2. `isSecureUrlV2(url)` Sử dụng **startsWith()**

Happy coding!

BÀI 5:

Rút trích domain từ địa chỉ email

Viết hàm `extractDomain(email)` nhận vào địa chỉ email, trả về phần domain sau ký tự @

Ví dụ:

- extractDomain('') --> ''
- extractDomain('alice@**gmail.com**') --> 'gmail.com'
- extractDomain('bob@**abc.com**') --> 'abc.com'

Viết hàm theo 2 hướng tiếp cận:

1. `extractDomainV1(email)` Sử dụng split
2. `extractDomainV2(email)` Sử dụng indexOf() và slice()

Happy coding!

BÀI 6:

Truy tìm mật mã

Viết hàm `findSecret(code)` để tìm ra chuỗi mật mã với quy tắc như sau.

Bỏ đi các ký tự **HOA** trong code, chuỗi còn lại chính là mật mã cần tìm.

Lưu ý: không dùng hàm replaceAll() và không dùng regular expression (regexp)

Ví dụ:

- findSecret('SUPERCODE') --> ''
- findSecret('SUPER**hello**CODE') --> 'hello'
- findSecret('**ea**ABFH**sy**UEYSJ**front**JSKJSH**end**') --> 'easyfrontend'

Happy coding!

Lưu ý: được phép dùng vòng **for** để duyệt chuỗi trong bài này.

BÀI 7:

Trả về full name khi biết first và last name

Viết hàm `getFullName(firstName, lastName)` nhận vào firstName và lastName và trả về chuỗi **fullName**.

Quy tắc để tạo chuỗi fullName như sau:

- firstName và lastName là optional (có thể không có)
- fullName không có khoảng trắng thừa ở đầu và cuối chuỗi
- firstName và lastName cần phải viết hoa chữ cái đầu tiên, chữ cái còn lại là viết thường

Ví dụ:

- getFullName('Alice') --> 'Alice'
- getFullName('Alice', '') --> 'Alice'
- getFullName('', 'Nguyen') --> 'Nguyen'
- getFullName('Bob', 'Tran') --> 'Bob Tran'
- getFullName('john', 'pHAm') --> 'John Pham'

Happy coding!

4 - OBJECT:

BÀI 1:

Viết hàm clone object nhưng không dùng spread operator

Viết hàm `function cloneObject(obj)` để clone một object **obj** truyền vào, và trả về là **một object mới** có đầy đủ các keys của object truyền vào.

Ví dụ:

```
const studentA = { name: 'Bob', math: 9 };
const studentB = cloneObject(studentA);

console.log(studentA === studentB); // should be false
console.log(studentB.name); // Bob
console.log(studentB.math); // 9
```

Lưu ý: Không sử dụng Object.assign() và spread operator

💡 BÀI 2:

Kiểm tra 2 objects có bằng nhau không?

Viết hàm `isEqual(obj1, obj2)` nhận vào 2 objects và trả về:

- **true** nếu số lượng keys của 2 objects bằng nhau, và giá trị của từng key cũng bằng nhau (dùng === để so sánh)
- ngược lại là false

Ví dụ:

- isEqual({}, {}) --> true
- isEqual({ name: 'Bob' }, { name: 'Alice' }) --> false
- isEqual({ name: 'Bob' }, { name: 'Bob' }) --> true
- isEqual({ name: 'Bob' }, { name: 'Bob', age: 18 }) --> false

Giả sử kiểu dữ liệu của các thuộc tính của cả 2 objects đều là kiểu dữ liệu primitive.

Happy coding!

5 - ARRAY

BÀI 1

Liệt kê các số trong khoảng [a, b]

Viết hàm `createArrayInRange(a, b)` để tạo ra mảng gồm các số nằm trong khoảng [a, b] (có bao gồm a và b)

Với a, b là các số thoả điều kiện sau: **-100 < a < b < 100**

Trả về là một array chứa các số trong khoảng từ a tới b.

Ví dụ:

- createArrayInRange(1, 5) --> [1, 2, 3, 4, 5]
- createArrayInRange(-2, 1) --> [-2, -1, 0, 1]

Hãy viết bài này bằng 2 cách:

1. Dùng for...i `createArrayInRangeV1(a, b)`
2. Dùng Array.from() `createArrayInRangeV2(a, b)`

Happy coding!

BÀI 2

Kiểm tra số nguyên tố

Viết hàm `isPrime(n)` nhận vào là số nguyên dương **(0 <= n < 1000)** và trả về kết quả **true/false** cho biết đó có phải là số nguyên tố hay không?

- Trả về true nếu là số nguyên tố
- Ngược lại, trả về false

Số nguyên tố là số chỉ chia hết cho 1 và chính nó (hay nói cách khác là chỉ có 2 ước số là 1 và chính nó) (Wikipedia)

Dãy số nguyên tố tham khảo: **2, 3, 5, 7, 11, 13, 17, ...**

- isPrime(2) --> true
- isPrime(3) --> true
- isPrime(4) --> **false** vì 4 ngoài chia hết cho 1, nó còn chia hết cho 2

Giải bài này bằng 2 cách:

1. Cách 1 là lặp từ 2 tới n - 1 để kiểm tra `isPrimeV1(n)`
2. Cách 2 là lặp từ 2 tới căn bậc 2 của n để kiểm tra `isPrimeV2(n)` (hãy suy nghĩ tại sao nhé)

BÀI 3

Liệt kê ước số của số nguyên dương n

Viết hàm `getDivisorList(n)` nhận vào số nguyên dương **(1 <= n <= 1000)** và trả về một mảng các ước số của n.

Ví dụ:

- getDivisorList(1) --> [1]
- getDivisorList(10) --> [1, 2, 5, 10]
- getDivisorList(12) --> [1, 2, 3, 4, 6, 12]

Hãy viết hàm này bằng 3 cách tiếp cận:

1. Sử dụng **for...i** từ 1 tới n `getDivisorListV1(n)`
2. Sử dụng Array.from() và filter() để lặp từ 1 tới n `getDivisorListV2(n)`
3. Sử dụng Array.from(), forEach() và sort(). Đồng thời chỉ lặp đến căn bậc 2 của n `getDivisorListV3(n)`

Happy coding!

BÀI 4

Kiểm tra số hoàn hảo

Viết hàm `isPerfectNumber(n)` để kiểm tra n có phải là số hoàn hảo hay không?

Với n thoả điều kiện 1 < n < 1000

Trả về **true** nếu đúng, ngược lại trả về **false**

Số hoàn hảo là số mà tổng của tất cả ước số (không tính chính nó, tức từ 1 đến n - 1) bằng chính nó.

Ví dụ: 6 = 1 + 2 + 3 (như vậy 6 là một số hoàn hảo)

Gợi ý: không nhất thiết phải chạy tới (n - 1) để tìm ra tất cả các ước số của n

BÀI 5

Biến đổi mảng với f(i) = f(i-1) + f(i + 1)

Viết hàm `transformNumbers(numberList)` để biến đổi các số hiện tại của mảng numberList thành các số mới theo công thức.

```
1.  f(i) = f(i - 1) + f(i + 1) với i là index
```

Tạm dịch nôm na là phần tử ở vị trí i sẽ bằng tổng của 2 phần tử bên cạnh.

Trường hợp đầu mảng và cuối mảng sẽ bằng phần tử liền kề.

Trường hợp mảng có ít hơn một phần tử thì sẽ giữ nguyên, không biến đổi.

Lưu ý: mảng trả về là một mảng mới, không phải là mảng truyền vào nhé!

Ví dụ

- transformNumbers([]) --> []
- transformNumbers([1]) --> [1]
- transformNumbers([5, 10]) --> [10, 5]
- transformNumbers([2, 4, 6, 8]) --> [4, 8, 12, 6] chú thích bên dưới

Đặt mảng đầu vào là a, mảng trả về là b, ta có:

- b[0] = a[1] = **4** (đầu mảng)
- b[1] = a[0] + a[2] = 2 + 6 = **8**
- b[2] = a[1] + a[3] = 4 + 8 = **12**
- b[3] = a[2] = **6** (cuối mảng)

Viết hàm này theo 3 cách khác nhau:

1. Dùng **for...i** `transformNumbersV1(numberList)`
2. Dùng **forEach()** `transformNumbersV2(numberList)`
3. Dùng **map()** `transformNumbersV3(numberList)`

Happy coding!

BÀI 6

Kiểm tra mảng có chứa số nguyên tố không?

Viết hàm `hasPrime(numberList)` nhận vào là một mảng số nguyên dương.

Trả về true nếu có ít nhất một số nguyên tố, ngược lại trả về false.

Gợi ý: có thể tận dụng lại hàm kiểm tra số nguyên tố đã làm ở bài tập trước đó.

Viết bằng 5 cách khác nhau:

1. Sử dụng for...i `hasPrimeV1(numberList)`
2. Sử dụng forEach() `hasPrimeV2(numberList)`
3. Sử dụng find() `hasPrimeV3(numberList)`
4. Sử dụng findIndex() `hasPrimeV4(numberList)`
5. Sử dụng some() `hasPrimeV5(numberList)`

Happy coding!

BÀI 7

Kiểm tra mảng có phải tất cả đều là số hoàn hảo không?

Viết hàm `isAllPerfectNumbers(numberList)` nhận vào mảng số nguyên dương.

Trả về **true** nếu tất cả đều là số hoàn hảo, ngược lại trả về **false**.

Ví dụ:

- isAllPerfectNumbers([]) --> false
- isAllPerfectNumbers([1, 6]) --> false vì 1 không phải là số hoàn hảo
- isAllPerfectNumbers([1, 2, 3]) --> false
- isAllPerfectNumbers([6]) --> true vì 6 là số hoàn hảo
- isAllPerfectNumbers([6, 28]) --> true vì 6 và 28 đều là số hoàn hảo

Viết hàm bằng 3 cách khác nhau:

1. `isAllPerfectNumbersV1(numberList)` Sử dụng **for...i** với cách tiếp cận là return false nếu phát hiện có một số ko phải là số hoàn hảo.
2. `isAllPerfectNumbersV2(numberList)` Sử dụng **reduce()** với cách tiếp cận là đếm số lượng số hoàn hảo, nếu nó bằng với length
   của array, nếu bằng thì return true, ngược lại return false
3. `isAllPerfectNumbersV3(numberList)` Sử dụng **every()**

BÀI 8

Tính trung bình cộng của các số chẵn trong mảng

Viết hàm `calcAvgOfAllEvenNumbers(numberList)` nhận vào là một mảng số nguyên dương.

Trả
 về một con số duy nhất cho biết trung bình cộng của tất cả số sẵn có trong mảng, nếu kết quả là số thực thì làm tròn về số nguyên gần nhất.

Ví dụ:

- calcAvgOfAllEvenNumbers(1) --> 0 vì dữ liệu đầu vào không hợp lệ
- calcAvgOfAllEvenNumbers([]) --> 0
- calcAvgOfAllEvenNumbers([1]) --> 0
- calcAvgOfAllEvenNumbers([1, 2]) --> 2
- calcAvgOfAllEvenNumbers([1, 2, 4]) --> **3** vì có 2 số chẵn 2 + 4 = 6, trung bình cộng lấy 6 / 2 = 3
- calcAvgOfAllEvenNumbers([1, 2, 4, 8]) --> **5** vì có 3 số chẵn 2 + 4 + 8 = 14, trung bình cộng lấy 14 / 3 = 4.6(6), làm tròn thành **5**

Happy coding!